ARINC-653 Inter-partition Communications and the Ravenscar Profile*

Jorge Garrido Juan Zamorano Juan A. de la Puente

Abstract

The ARINC-653 standard is often used to build mixed-criticality systems, using a partitioned architecture. Inter-partition communication is carried out by means of a message-passing mechanism based on ports. The standard includes an API for Ada, but the implementation semantics of operation ports is not fully defined. Furthermore, the API was defined for the Ada 95 standard, and therefore does not take into account the enhancements to the real-time features of the language that have been incorporated in the 2005 and 2013 standards, most notably the Ravenscar profile. This paper is aimed at clarifying the implementation of ARINC communication ports in Ada and the Ravenscar profile. ARINC communication ports are analysed, and their compatibility with the Ravenscar profile is assessed. A new API that can be used with the profile is defined, and a pilot implementation is introduced.

1 Introduction

Partitioning is a convenient approach for developing mixed-criticality systems on a single hardware platform. Under this approach, a system is divided into a set of logical partitions, which are isolated from each other both in the temporal and spatial domains. Subsystems with different levels of criticality are allocated to different partitions, and failures in low-criticality components cannot propagate to high-criticality ones. Critical subsystems can thus be certified in isolation to the required level, provided that the partitioning kernel is also certified to that level, but the validation of lower-criticality subsystems can be greatly simplified.

The ARINC-653 standard has been used for some time to build partitioned systems, mainly in the aerospace domain. It defines an Integrated Modular Avionics (IMA) architecture, based on a partitioning kernel with a programming interface called APEX (Application Executive). The kernel executes partitions based on a static scheduling plan, whereas a partition OS can be used to schedule tasks within each partition. Inter-partition communication is based on message passing through message ports.

This paper is aimed at clarifying the implementation of ARINC communication ports in Ada, specifically using the Ravenscar profile. The ARINC standard defines an API for Ada83 and Ada95, and a set of data types to be used with communication primitives, but the semantics of the interfaces is not clearly defined. Furthermore, it does not mention the Ravenscar profile as it was not part of the Ada standard until 2005.

The rest of the paper is organized as follows. The ARINC communication ports are analysed in section 2. Guidelines for a Ravenscar-compatible API are defined in section 3. A pilot implementation of the API on XtratuM, an open-source partitioning kernel based on virtualization techniques, is presented in section 4. Finally, section 5 contains some conclusions and plans for future work.

2 ARINC-653 Inter-partition Communication

2.1 Channels and ports

The ARINC-653 standard [1] defines an inter-partition communication (IPC) mechanism that can be used to exchange messages between ARINC partitions running on the same computer board. It can also be used with ARINC partitions or non-ARINC applications running on different computer boards. It is worth noting that communication is carried out between partitions, not tasks or processes within a partition.

A message is defined as a contiguous block of data of finite but variable length. Messages are exchanged through channels. A channel is a logical link between a source partition and one or more destination partitions. In this way, partitions can send and receive messages through multiple channels via defined access points, called ports. Channels and ports are defined as part of the system configuration activities, and therefore source and destination ports and other characteristics of each channel cannot be changed at run-time.

The standard does not define the underlying transport mechanism, but it is assumed that the messages leave the source port and reach the destination ports in the same order. Therefore, the transport mechanism is transparent to the applications, allowing ARINC 653 applications to communicate in the same way regardless of whether they run on the same or on different computer boards.

There are two different modes of message transfer: sampling mode and queuing mode. Each individual port can be configured to operate in a specific mode, and thus it is not prohibited that ports operating in different modes can be connected to the same channel.

Ports have attributes that are used to control and maintain the operation of ports and portions of channels located within the same computer board. Some of the attributes, such as the partition identifier, the port name, the mode of transfer, the direction, and the maximum message size, are common to all ports. Other attributes, such as the refresh period, the maximum number of messages, and the process queue discipline, are specific to ports of a particular kind. The maximum message size and the maximum number of messages define the storage amount in the memory space of the partition which is needed to buffer messages within a port.

2.2 Sampling mode

The sampling mode is intended to transport messages with identical structure but varying, updated data. As a result, only one message is kept at sampling ports. A message remains in the source port until it is transmitted by the channel or it is overwritten by a new occurrence of the message. In a similar way, an arriving message overwrites the latest message at the destination port. In this way, destination partitions can only access the latest message. As messages are atomic entities, a partial message will not be delivered to the destination port.

A refresh period can be specified as a port attribute that indicates the maximum acceptable age of a valid message. In this way, a validity status is returned when reading messages. If the message is read after its validity period expires, a warning is issued, although the message can still be retrieved.

2.3 Queuing mode

The queuing mode is intended to transport messages with uniquely different data, and therefore a message is not allowed to overwrite previous messages transmitted on the same channel. As a result, queuing ports have to buffer multiple messages in message queues. Messages sent through a queuing port are stored in the associated message queue until they are transmitted, and messages arriving at a queuing port are stored in the corresponding message queue until they are retrieved by the destination partitions.

A maximum number of buffered messages must be specified for each message queue. Overflows are handled by the ports. Application tasks may be blocked on a full message queue when sending a message, or

on an empty message queue when receiving a message. A time-out can be specified for transmit or receive operations, and therefore the maximum blocking time can be limited. A zero waiting time can be used to invoke a non-blocking operation. Tasks waiting on a queuing port can be arranged by FIFO or priority order.

3 A Ravenscar API for ARINC IPC

Ada 83 and Ada 95 bindings to the APEX services can be found in appendix D of the ARINC standard [1]. The bindings include definitions for constants and types, as well as the required subprograms. It must be noticed that the bindings are intended to be used without the Ada concurrency facilities. APEX provides its own services for creating, scheduling, communicating and synchronizing processes within a partition, and Ada applications are supposed to use these services rather than native Ada tasking.

Such scenario is now outdated, as Ada tasking and the Ravenscar profile provide a much more convenient framework for building real-time systems. What is needed is an API for APEX IPC that can be used by a Ravenscar kernel running as a host operating system in an ARINC partition. We shall assume that tasking is implemented by the Ada run-time, and thus APEX process services are not used. Therefore, the API will address only APEX services related to sampling and queuing ports.

The original Ada 95 interface specification of sampling and queuing ports in the standard [see 1, ap. D2] is compatible with the Ravenscar profile as long as it does not appear violate any Ravenscar restriction (see listing 1 for a sample). Indeed, some types and constants could be better implemented with more appropriate Ada 95 features such as `Streams`. The original reason for not using such abstract features seems to be trying to keep the interface as close as possible to the older Ada 83 specification.

A key issue is found, however, related to the implementation of APEX queuing ports, specifically regarding time-outs and queuing discipline, as these mechanisms are restricted by the Ravenscar profile. It could be argued that the implementation of time-outs and queues is outside the scope of the Ravenscar kernel, as it is provided by the underlying APEX executive. However, the schedulability of the partition may be compromised if the APEX services does not comply with the Ravenscar computational model, as is the case here. Indeed, tasks may undergo unbounded blocking time when sending or receiving messages through queuing ports. Moreover, in the case of priority-ordered queues, there is an additional problem as the APEX only deals with APEX processes and priorities, and it has no notion of Ada tasks and priorities.

A trivial solution would be to provide a binding without the time-out parameter, and always pass a zero-valued time-out parameter to the APEX subprogram. In this way, time-outs and task queues could be avoided in Ravenscar applications. However, these features were kept out of the Ravenscar profile in order to achieve a simple run-time, and to enable the use of static temporal analysis techniques [4]. Therefore, the fact that the ARINC operating system supports them, even if not used in Ravenscar partitions, may compromise the certification of these partitions and even the whole system.

Fortunately, part 4 of the ARINC standard [2] defines a subset of required services with aims similar to the Ravenscar profile. In this subset, the IPC services are restricted to non-blocking mode of operation in order to remove waiting queues and simplify process management. Moreover, the queuing discipline attribute of a queuing port is ignored.

Furthermore, this subset only allows up to two processes (one periodic and/or one aperiodic process) per partition. The periodic process is intended to provide the functionality associated with the partition, and is recommended to be assigned the highest priority in the partition. The aperiodic process is intended to act as a background process with a lower priority. This simplifies the mapping of Ravenscar partitions to APEX, as all the application code can be mapped to the periodic process, including Ada tasks that are scheduled by the Ravenscar run-time.

Listing 2 shows the modified API taking into account the above considerations. Notice the use of Ada streams to represent messages.

Listing 1: Extract of APEX IPC Ada 95 bindings

```
package APEX.Sampling_Ports is
   ...
   procedure Write_Sampling_Message
    (Sampling_Port_Id : in Sampling_Port_Id_Type;
     Message_Addr     : in Message_Addr_Type;
     Length           : in Message_Size_Type;
     Return_Code      : out Return_Code_Type);

   procedure Read_Sampling_Message
    (Sampling_Port_Id : in Sampling_Port_Id_Type;
     Message_Addr     : in Message_Addr_Type;
     Length           : out Message_Size_Type;
     Validity         : out Validity_Type ;
     Return_Code      : out Return_Code_Type);
    ...
end APEX.Sampling_Ports;

package APEX.Queuing_Ports is
   ...
   procedure Send_Queuing_Message
      (Queuing_Port_Id : in Queuing_Port_Id_Type;
       Message_Addr    : in Message_Addr_Type;
       Length          : in Message_Size_Type;
       Time_Out        : in System_Time_Type;
       Return_Code     : out Return_Code_Type);

   procedure Receive_Queuing_Message
      (Queuing_Port_Id : in Queuing_Port_Id_Type;
       Time_Out        : in System_Time_Type;
       Message_Addr    : in Message_Addr_Type;
       Length          : out Message_Size_Type;
       Return_Code     : out Return_Code_Type);
   ...
end APEX.Queuing_Ports;
```

Listing 2: Simplified APEX IPC bindings for Ada Ravenscar profile

```
package IPC.Sampling_Ports is
   ...
   procedure Write_Sampling_Message
    (Sampling_Port_Id : in  Sampling_Port_Id_Type;
     Message          : in  Stream_Element_Array;
     Return_Code      : out Return_Code_Type);

   procedure Read_Sampling_Message
    (Sampling_Port_Id : in  Sampling_Port_Id_Type;
     Message          : out Stream_Element_Array;
     Return_Code      : out Return_Code_Type);
    ...
end IPC.Sampling_Ports;

package IPC.Queuing_Ports is
   ...
   procedure Send_Queuing_Message
     (Queuing_Port_Id : in  Queuing_Port_Id_Type;
      Message         : in  Stream_Element_Array;
      Return_Code     : out Return_Code_Type);

   procedure Receive_Queuing_Message
     (Queuing_Port_Id : in  Queuing_Port_Id_Type;
      Message         : out Stream_Element_Array;
      Return_Code     : out Return_Code_Type);
   ...
end IPC.Queuing_Ports;
```

4 The Ravenscar IPC API for XtratuM

XtratuM is an open source hypervisor [5, 8] that provides an APEX interface, including the IPC subset defined in part 4 of the ARINC 653 standard [2]. It is written in C, and is available for LEON and PC-compatible systems. The ORK kernel has been ported to XtratuM for LEON2 systems [6], and later upgraded to LEON3 monocore and multicore systems. The resulting open partitioning platform has been used for demonstrations in the EagleEye [3] and MultiPARTES [11] projects. A Ravenscar-compatible IPC interface following the above explained guidelines was developed to be used in these projects.

Periodic messages are commonly used in real-time systems, e.g. sensor data that are periodically sent to be used for a control algorithm. This kind of messages are properly supported by ARINC IPC sampling channels. The last valid data sample is used by the application, although samples may occasionally be overwritten due to response, activation, or transmission jitter.

Aperiodic or sporadic messages, which are also often needed, can also be exchanged on message channels, but in this case the receiving tasks must periodically poll the respective ports. This approach is not convenient as it is known that the period of the polling task depends on the minimum inter-arrival time of the messages and the deadline of the corresponding activity [9]. The resulting period is shorter than the minimum inter-arrival time of an event-activated sporadic task. Therefore, a periodic polling task will yield more interference on low priority tasks than a sporadic task. Consequently, it is better to use some kind of notification mechanism to detect when a sporadic message has arrived and use it to activate a sporadic task.

Virtual interrupts are customarily used in partitioned kernels to mask actual hardware interrupts, which are handled by the kernel and notified to partitions as software events. XtratuM provides such a mechanism for hardware interrupts, and also uses it to provide an additional set of virtual interrupts that are used to notify scheduling and other kernel events. In particular, there are two virtual interrupts that are used to notify partitions of the availability of new messages in sampling or queuing destination ports. It must be noticed, though, that the support for the additional virtual interrupts requires adding the corresponding entries in the virtualized interrupt table, which may make it unpractical to provide a different virtual interrupt for each possible port in a partition. Another issue may be a loss of portability to other ARINC 653 platforms as virtualized interrupts are not part of the standard. However, the advantage of using sporadic tasks to retrieve non-periodic messages has motivated the choice of this mechanism to implement the ORK binding to XtratuM.

Listing 3 shows a sporadic task pattern using the XtratuM API. The code is similar to common Ravenscar stereotypes for sporadic tasks and device drivers [7, 10]. The task waits until a new message is available on the sampling port, and then the message is retrieved. The protected procedure `Signal` handles the `Sampling_Port` virtual interrupt. In this case, the interrupt handler just opens the barrier to release the task. It must be said that the package `Ada.Interrupts.Names` not only declares the Ada names of interrupts but also their corresponding priorities.

In practice, having only one interrupt per port type may compromise the support of multiple ports on a single partition. The corresponding interrupt is signalled when a message is received on a specific port, thus activating the aperiodic task. To avoid propagating this activation to all the tasks waiting for the reception of a message, a higher level logic has to be added. For the IPC API on Xtratum, a subscription-based protocol has been implemented. In this protocol, tasks can be associated with specific ports and blocked on protected object entries associated to that port. In this way, entries are only unlocked by the sporadic task at the reception of messages at the specified port.

From the real-time point of view, this protocol has two implications. On one hand, this approach only activates a task when it has a message to process. This avoids unnecessary context switches due to tasks activations with invalid data. On the other hand, the response time of the sporadic task is linearly increased by the number of ports. When an interrupt is received, the task has to check all ports for new messages. However, the overhead generated by each additional port can be easily measured. Then, a maximum number

Listing 3: Sporadic task pattern.

```
protected Sampling_Port_Interrupt is
   pragma Priority (Ada.Interrupts.Names.Sampling_Port_Priority);
   entry Wait;
   procedure Signal;
   pragma Attach_Handler (Signal, Ada.Interrupts.Names.Sampling_Port);
private
   Signalled : Boolean := False;
end Sampling_Port_Interrupt;
...
task body Reader is
   ...
begin
   loop
         Sampling_Port_Interrupt.Wait;
      XM_Read_Sampling_Message(Port, Message, Flags, Result);
      ...
   end loop;
end Reader;
```

of ports per partition can be defined to bound the sporadic task response time according to the temporal requirements of the partition.

5 Conclusions and future work

The implications of using the ARINC 653 IPC mechanisms in Ada real-time systems with the Ravenscar profile have been discussed, and guidelines for a Ravenscar-compliant API have been developed. A pilot implementation on an open-source partitioning kernel has been built, and guidance on using the IPC API for implementing periodic and sporadic message-driven tasks has been provided.

Planned future work includes elaborating more detailed patterns for automatic code generation and integration with temporal analysis tools.

References

[1] ARINC. *Avionics Application Software Standard Interface: ARINC Specification 653 Part 1, Required Services*. ARINC, November 2010.

[2] ARINC. *Avionics Application Software Standard Interface: ARINC Specification 653 Part 4, Subset Services*. ARINC, June 2012.

[3] V. Bos, P. Mendham, P. Kauppinen, N. Holsti, A. Crespo, M. Masmano, J. de la Puente, and J. Zamorano. Time and space partitioning the EagleEye Reference Mission. In *Data Systems in Aerospace — DASIA 2013*, Porto, Portugal, 2013.

[4] A. Burns, B. Dobbing, and G. Romanski. The Ravenscar tasking profile for high integrity real-time programs. In L. Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, volume 1411 of *Lecture Notes in Computer Science*, pages 263–275. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64536-8. doi: 10.1007/BFb0055011.

[5] A. Crespo, I. Ripoll, and M. Masmano. Partitioned embedded architecture based on hypervisor: The XtratuM approach. In *European Dependable Computing Conference — EDCC 2010*, pages 67 –72, april 2010.

[6] A. Esquinas, J. Zamorano, J. A. de la Puente, M. Masmano, I. Ripoll, and A. Crespo. ORK+/XtratuM: An open partitioning platform for Ada. In A. Romanovsky and T. Vardanega, editors, *Reliable Software Technologies — Ada-Europe 2011*, number 6652 in LNCS, pages 160–173. Springer-Verlag, 2011.

[7] J. López, Á. Esquinas, J. Zamorano, and J. A. de la Puente. Experience in programming device drivers with the Ravenscar profile. *Ada User*, 31(2), June 2010.

[8] M. Masmano, I. Ripoll, A. Crespo, and J.-J. Metge. XtratuM: a hypervisor for safety critical embedded systems. In *11th Real-Time Linux Workshop*, Dresden. Germany., 2009.

[9] A. K. Mok. The design of real-time programming systems based on process models. In *IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, 1984.

[10] J. Pulido, J. A. de la Puente, M. Bordin, T. Vardanega, and J. Hugues. Ada 2005 code patterns for metamodel-based code generation. *Ada Letters*, XXVII(2):53–58, August 2007. Proceedings of the 13th International Ada Real-Time Workshop (IRTAW13).

[11] E. Salazar, A. Alonso, and J. Garrido. Mixed-criticality design of a satellite software system. In E. Boje and X. Xia, editors, *Proc. 19th IFAC World Congres*, pages 12278–12283. IFAC-PapersOnLine, 2014.